Year 4 — Practical Numerical Analysis

Based on lectures by Dr Kathryn Gillow
Notes taken by James Arthur

Michaelmas 2022

These notes are not endorsed by the lecturers, and I have modified them (often significantly) after lectures. They are nowhere near accurate representations of what was actually lectured, and in particular, all errors are almost surely mine (especially the typos!).

Contents

1 Introduction

Problem Sheet by Tuesday at 12 noon. Thursday on the problem sheets from week 2. TA is Robert McDonald. We will study,

- Rootfinding
- ODEs
  - Euler Schemes
  - Ruga Kutta
  - Linear multistep
- Parabolic PDEs - Heat Equation

1.1 Root Finding (and optimisation)

The idea is simple. Find some x such that $f(x) = 0$. We are being vague on purpose. The stating is simple, the solving isn't simple. Take the cubic, this isn't simple. Then for the quintic, then it's only has closed form when its Galois group is solvable. These closed forms, may not exist, be unstable, require evaluation of special function. Hence we need numerical methods.

The relation to optimisation is obvious. The max or min occurs at a turning point. That is,

$$f_i := \frac{\partial g}{\partial x_i} = 0$$

and then we have a root finding problem.

1.1.1 Univariate Root Finding

In the 1D case, given $f : [a, b] \to \mathbb{R}$ then find $c \in [a, b]$ such that $f(c) = 0$. The bisection algorithm uses the intermediate value theorem (See Analysis / Topology). Thus to find a root of $f(c)$ we find a and b such that $f(a)$ and $f(b)$ have opposite signs. Then let $c = (a + b)/2$. Then compute $f(c)$, check and repeat.

We need to decide when to terminate the algotithm, the normal are,

- $|b - a| < \text{tol}$,
- $|f(c)| < \text{tol}$.

For convergence, since $c \in [a, b]$ the maximum error is $b - a$. Since the step halves, the error at step n,

$$|c_n - c| \leq \frac{b-a}{2^n}.$$

Thus to achieve an accuracy of tolerance requires,

$$n \geq \frac{\log(b-a) - \log(\text{tol})}{\log(2)}$$

steps of bisection algorithm.

Here is some pseudocode for this algorthm,
So can we do better? Well yes,

1.1.2 Regular Falsi

This algorithm finds a clever guess. We are going to approximate f on $[a, b]$,

$$p_1(x) = \frac{x-a}{b-a}f(b) + \frac{b-x}{b-a}f(a)$$

Then $p_1(x)$ has a root at,

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

and then we use this in the bisection algorithm.

This fails for functions like $f(x) = x^{10} - 1/10$. There is a fix,

$$c = \frac{af(b)/2 - bf(a)}{f(b)/2 - f(a)}$$

This is known as the illinois algorithm.

1.1.3 Newton Raphson

Suppose we approximate $f(x)$ by $g(x)$ which is the truncated Taylor series of f about the point x_k,

$$g(x) = f(x_k) + (x - x_k)f'(x_k)$$

Then $g(x)$ has a root at,

$$c = x_k - \frac{f(x_k)}{f'(x_k)}$$

Thus the newton raphson method requires an intial guess and then iterates,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

until convergence is achieved. This is quicker, quadratic, if we have a guess sufficiently close.

We can variate this to secant method. The iterate is,

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k+1}}{f(x_k) - f(x_{k-1})}$$

This avoids the need for $f'(x_k)$. We also have damped newtons,

$$x_{k+1} = x_k - \sigma_k\frac{f(x_k)}{f'(x_k)}$$

for some $\sigma_k \in (0, 1]$. A third is Halley's method,

$$x_{k+1} = x_k - \frac{2f(x_k)f'(x_k)}{2(f'(x_k))^2 - f(x_k)f''(x_k)}$$

This works if the roots x_* satisfies $f'(x_*) \neq 0$ and is then Newtons method applied to the function $f(x)/\sqrt{|f'(x)|}$.

1.1.4 Polynomial Root Finding

Suppose we want to find the root of a Polynomial,

$$p(z) = \sum_{i=0}^{n} a_i z^i$$

where $a_n = 1$. Methods based on bisection will only find real roots, one at a time. Methods based on newtons will find roots one at a time. With a complex initial guess such methods will find complex roots.

In order to find all the roots at once, we can define the companian matrix $C \in \mathbb{C}^{n\times n}$ as,

$$C = \begin{pmatrix} 0 & & & -a_0 \\ 1 & \ddots & & \vdots \\ & \ddots & 0 & -a_{n-2} \\ & & 1 & -a_{n-1} \end{pmatrix}$$

Then we can consider $zI - c$,

$$zI - C = \begin{pmatrix} z & & & a_0 \\ -1 & \ddots & & \vdots \\ & \ddots & z & a_{n-2} \\ & & -1 & z + a_{n-1} \end{pmatrix}$$

and then we can see that $\det(zI - C) = p(z)$. Hence the roots of $p(z)$ is equivalent to finding the eigenvalues of matrix C. This is what matlab `roots` uses this approach. You may think this is circular, but we can find eigenvalues in other ways, like the QR algorithm.

1.1.5 Multivariate Root Finding

Now we want to find x such that $f(x) = 0$ for $f : \mathbb{R}^n \to \mathbb{R}^n$. Algorithms based on bisections aren't extendable to higher dimensions. Newton based algorithms are easier to extend. Again we approximate $f(x)$ by the first few terms,

$$f(x + \delta x) \approx f(x) + J(x)\delta(x)$$

where J is the jacobean. Then we need to find the newton iterate,

$$J(x)\delta x = -f(x)$$

such that,

$$J(x_k)(x_{k+1} - x_k) = -f(x_k)$$

We can also write the iterate as,

$$x_{k+1} = x_k - (J(x))^{-1} f(x_k)$$

thus damped can be written as,

$$x_{k+1} = x_k - \sigma_k (J(x))^{-1} f(x_k)$$

Here's the kicker. We don't really know how many solutions a root finding algorithm has. We need to look more closely. Hence we bring the idea of newton fractals. For example, suppose we have $f(z) = z^3 - 1$. This has three roots, $z = 1, (-1 \pm \sqrt{3}i)/2$. By writing $z = x + iy$ and equating real and imaginary parts. We get,

$$\begin{aligned} x^3 - 3xy^2 - 1 &= 0 \\ 3x^2 y - y^3 &= 0 \end{aligned}$$

which can be solved using Newtons Method. For each point for $(x, y) \in [-2, 2]$ we compute a solution and record what root it converges to. A newton fractal is what this converges to.